SUMMARY:
We plan to create an optimized surface detection and renderer algorithm to render objects on a seen plane in real time. We are initially going to create this optimized implementation on the GPU. If we succeed, we will also complete one for multi-core CPU platforms, and then perform an analysis on both systems’ performance characteristics to compare and contrast them. While we plan to initially implement this on a real-time computer, we hope to also be able to create an application runnable on a mobile devices as well.

BACKGROUND:
Our algorithm can be broken up into three distinct portions. The first part is the actual detection of a flat plane in a given image that could be drawn upon. The second would be the tracking of the said plane over real-time applications. The final part would be the rendering of a given item on the given plane. Each of these parts would be initially implemented sequentially, and then done so in parallel. For the detection of a plane, we’re planning on using OpenCV to extract features, compute the homographies of the features using the RANSAC method, and create our planed surface to sketch on (the plane detection of our algorithm). To continually track this plane, we’re planning on utilizing a Lucas-Kanade Tracker to follow our object once it’s been instantiated. Finally, for the rendering of our object, we’re planning on using OpenGL to render these images on the planes. Parallelization could definitely improve parts of our algorithm! For example, the feature extraction and matching between our imposed plane and the image we gather could be parallelized over multiple processors.

THE CHALLENGE:
This algorithm provides a lot of challenges that we hope to overcome. First among them is the transition from sequential code to parallel code. There are some steps (such as the detection of a plane), which seem very easy to do sequentially. The calculation of an available plane uses arbitrarily swaths of memory, and using a sequential algorithm places all of that needed memory in a single processor. However, when parallelizing the task, the given plane we’re looking for could have been processed by multiple processors, making this problem of workload and memory sharing
an issue. This makes mapping the workload of our space extremely challenging, as we need to balance utilizing multiple processors with divvying up the workspace efficiently.

RESOURCES:
To test our implementation, we will be harvesting video footage of real-time video and then running the code on the late-days cluster. We currently do not plan to use a code base, and are planning on starting from scratch. So far, we’ve looked into using OpenCV for an object-detection library. We’re planning on using this to extract features, compute homographies using the RANSAC method, and create our planed surface to sketch on (the plane detection of our algorithm), as well as the implementation of our Lucas-Kanade Tracker. The OpenGL functions that we plan to use take care of a large part of the rendering we have, and we can set the center point of this rendering to be the center point of the plane we have.

GOALS AND DELIVERABLES.
In order to achieve a “successful project”, we must implement the baseline algorithm described above in parallel that can run efficiently on a GPU. This means writing a parallel plane detection algorithm, which can be tracked by a parallel tracking algorithm, and finally be drawn upon with a parallel rendering algorithm. In order to go above and beyond the scope of our project, we could implement this algorithm on multiple CPUs and perform analytics to compare how the program runs on both platforms. We could also try porting our code onto a mobile device and seeing whether or not it functions properly and efficiently. In case the work goes slower than initially anticipated, we could probably utilize a sequential portion of the algorithm such as plane detection or tracking.

We hope to see quite a bit of speedup between the sequential and parallel versions of our code! We can't come up with a decent goal as of now, but we believe it’s fair to assume at least 5x speedup.

To perform a demo of our project during the poster session, we could take real time video, upload it to a late-days cluster, perform our algorithm, and then get back a video of the object being rendered on a plane in real time! We will be able to show up speedup graphs as well.

PLATFORM CHOICE.
We plan to use C++, as it has the support needed for OpenCV, and we have experience creating parallelized code in the language.

SCHEDULE.
Current Schedule:

<table>
<thead>
<tr>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>Parallel</td>
<td>Sequential</td>
<td>Parallel</td>
<td>Sequential</td>
<td>Parallel</td>
</tr>
<tr>
<td>planar finding</td>
<td>planar finding</td>
<td>planar tracking</td>
<td>planar tracking</td>
<td>Rendering</td>
<td>Rendering</td>
</tr>
</tbody>
</table>